Duccio Rocca
ID = 922254031
April 19th, 2022

Assignment 3 Documentation

Github Repository

Assignment four repository.
or
https://github.com/sfsu-csc-413-fall-2022-roberts/assignment-4---interpreter-RINO-GAELICO

Project Introduction and Overview

This project required me to complete an Interpreter starting from a skeleton code. It required me to implement some classes from scratch (E.g. all the bytecode classes) and to complete others starting from a template (for example Program class).

The assignment was divided into two parts, the first part required me to implement the various part of the Interpreter so that could read bytecode and execute it. The second part required me to implement a method to output the bytecode and runtime stack giving particular importance to the format of the output.

Execution and Development Environment

As IDE I used Visual Studio Code on my MacBook Pro and I tested in both IDE and terminal through the command line.

JDK is Openjdk, version "17.0.2" 2022-01-18. The OpenJDK Runtime Environment is build 17.0.2+8-86, and OpenJDK 64-Bit Server VM is build 17.0.2+8-86 (mixed mode, sharing).

Assumptions

I assumed that the bytecode programs used are generated correctly, and therefore should not contain any syntactical errors.

I also assumed that the Interpreter class worked correctly and didn’t require any correction.

Scope of Work

Task		Completed
Provide implementations for each of the byte codes discussed in class		X
	HALT	X
	POP	X
	FALSEBRANCH	X
	GOTO	X
	STORE	X
	LOAD	X
	LIT	X
	ARGS	X
	CALL	X
	RETURN	X
	BOP	X
	READ	X
	WRITE	X
	LABEL	X
The byte code classes should		X
	contain parameterless constructors	X
	should contain any member data necessary for that byte code to function properly.	X
	provide some method that allows for the creator of that byte code to provide any data necessary for that byte code to function properly. (`init()`)	X
	Byte codes must be placed in a sub package	X
Implement a parent abstract base class		X
CodeTable class that stores the mapping between byte codes and the class that implements that byte code behavior		X
ByteCodeLoader that reads each line from the specified byte code file		X
	is responsible for creating byte code instances	X
Program class that holds the byte codes generated by the ByteCodeLoader		X
	Program class must provide a method to resolve symbolic addresses to a numeric byte code address	X
RuntimeStack to manage the runtime stack.		X
	The RuntimeStack must record the actual data stored in the current runtime stack in a runStack Vector	X
	The RuntimeStack must track the activation records in a framePointers Stack.	X
Remove all debug statements		X
DUMP bytecode Class		X
	The ON flag will set an interpreter switch that will cause dumping AFTER execution	X
	Byte code instruction that is output should be left aligned in 25 columns, and any additional information should begin at column 26	X
	Recursive method to adjust position of children if necessary	X
Special treatment for the following bytecodes		X
	LIT	X
	LOAD	X
	RETURN	X
	CALL	X

	STORE	X
A dump() method should be added to the RuntimeStack to allow the object to dump the current state of the stack		X
Check for stack underflow errors, or popping past a frame boundary.		X
	Pop method perform the check	X

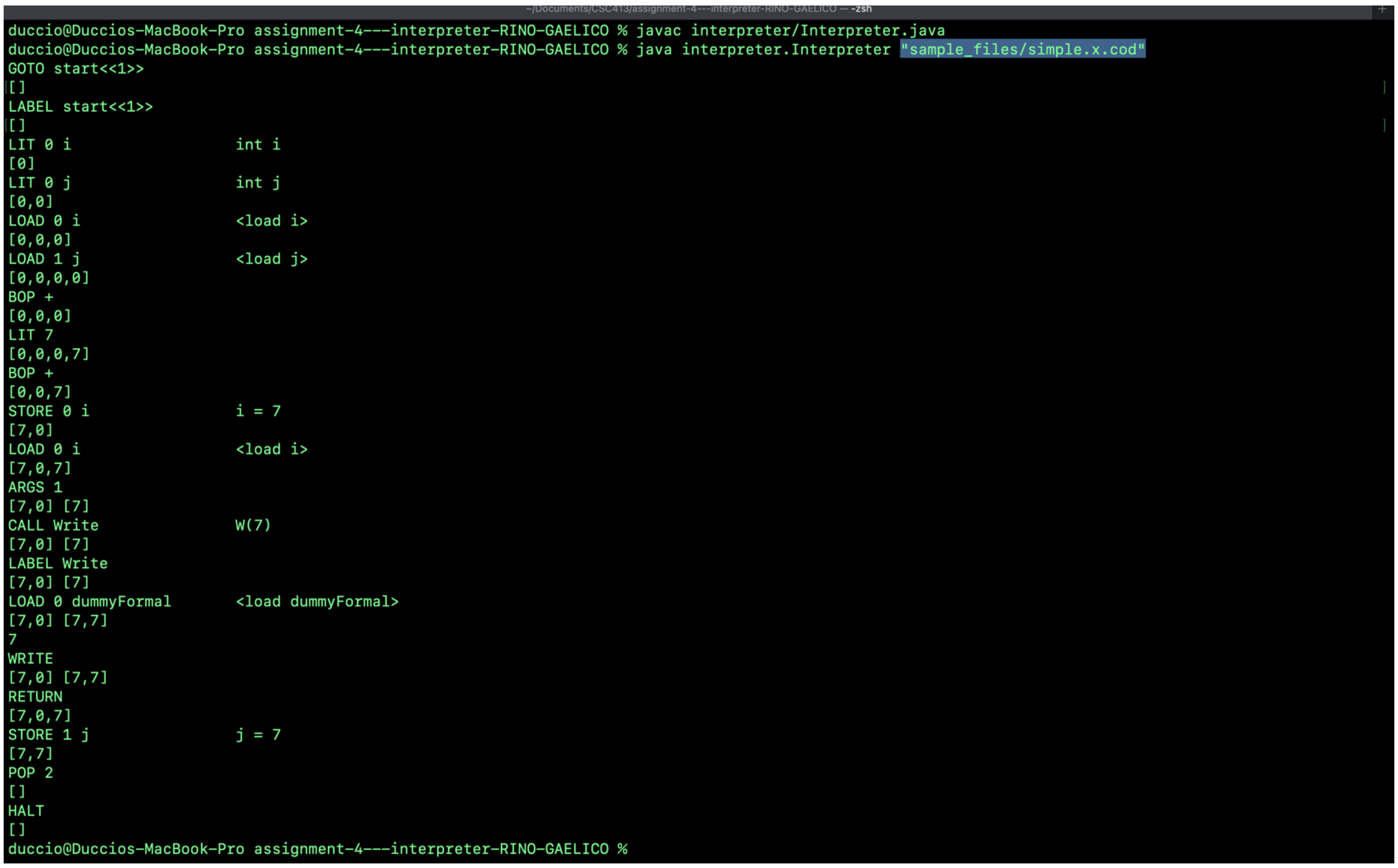
Compilation Result

First, I compiled all the bytecodes in the bytecode package with the following command:

```
javac interpreter/bytecode/*.java
```

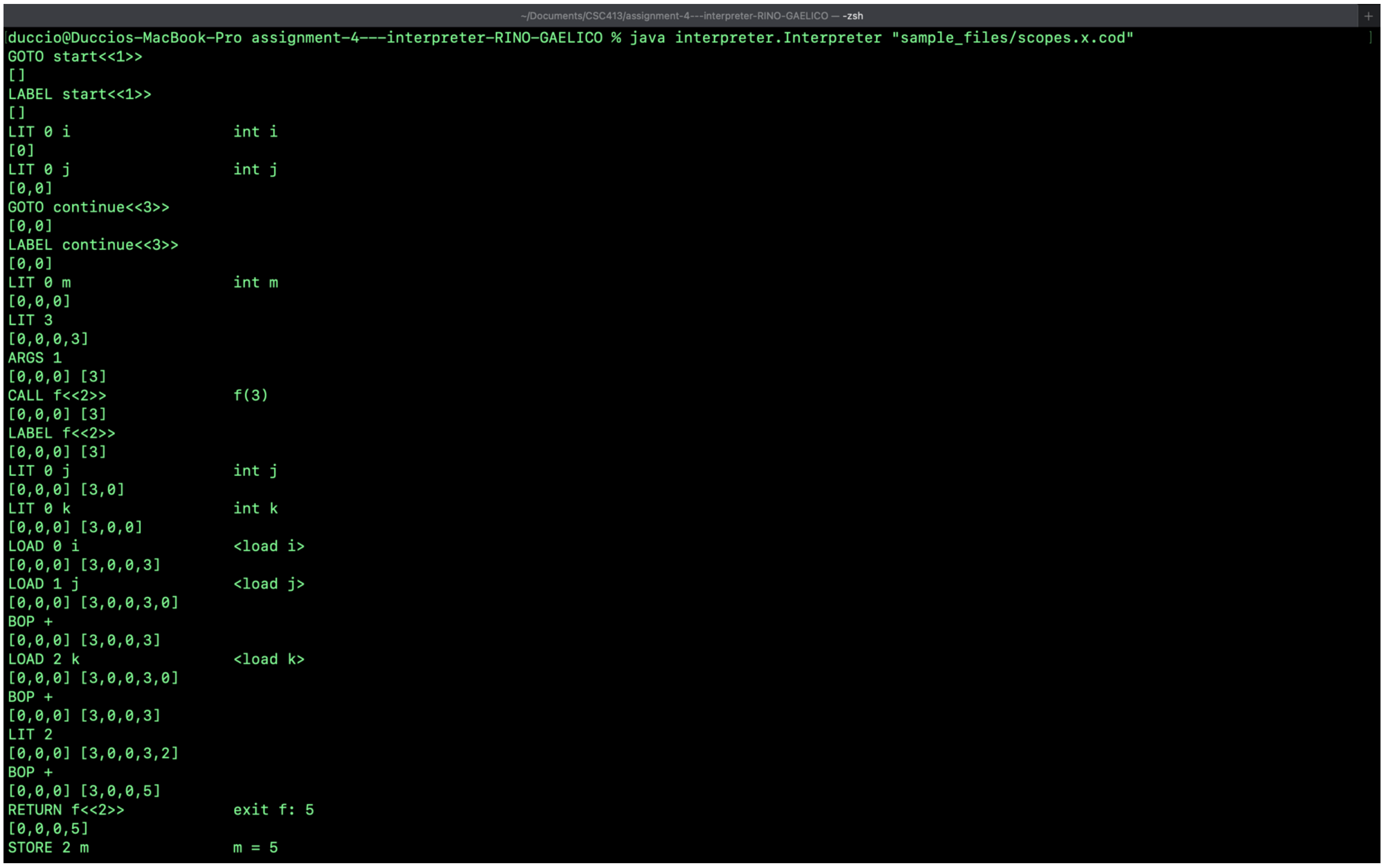
I then compiled and ran the Interpreter class with the commands shown below, using simple.x file as a test. No error messages or warnings were displayed, and the application ran as expected. I modified the bytecode files so that DUMP was On from the beginning to end.

```
javac interpreter/Interpreter.java
java interpreter.Interpreter "sample_files/simple.x.cod"
```

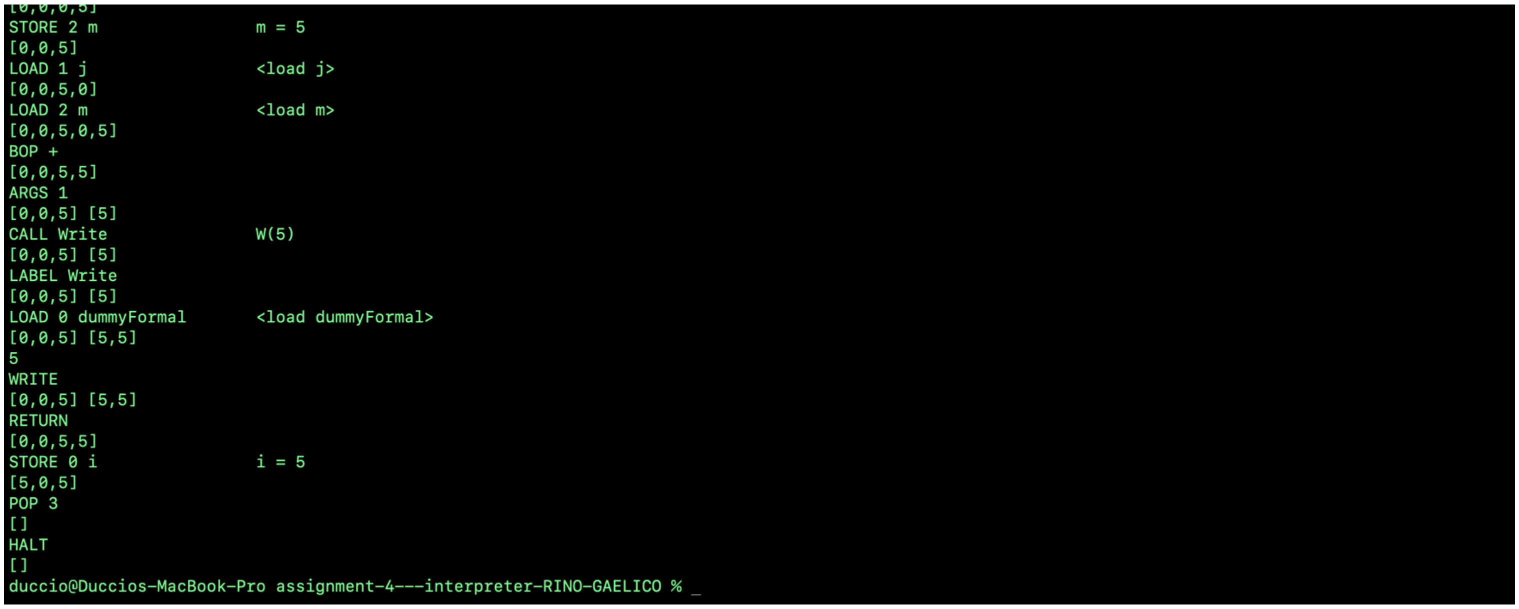


```
~/Documents/CSC413/assignment-4---interpreter-RINO-GAELICO — -zsh
duccio@Duccios-MacBook-Pro assignment-4---interpreter-RINO-GAELICO % javac interpreter/Interpreter.java
duccio@Duccios-MacBook-Pro assignment-4---interpreter-RINO-GAELICO % java interpreter.Interpreter "sample_files/simple.x.cod"
GOTO start<<1>>
[]
LABEL start<<1>>
[]
LIT 0 i                  int i
[0]
LIT 0 j                  int j
[0,0]
LOAD 0 i                 <load i>
[0,0,0]
LOAD 1 j                 <load j>
[0,0,0,0]
BOP +
[0,0,0]
LIT 7
[0,0,0,7]
BOP +
[0,0,7]
STORE 0 i                i = 7
[7,0]
LOAD 0 i                 <load i>
[7,0,7]
ARGS 1
[7,0] [7]
CALL Write               W(7)
[7,0] [7]
LABEL Write
[7,0] [7]
LOAD 0 dummyFormal       <load dummyFormal>
[7,0] [7,7]
7
WRITE
[7,0] [7,7]
RETURN
[7,0,7]
STORE 1 j                j = 7
[7,7]
POP 2
[]
HALT
[]
duccio@Duccios-MacBook-Pro assignment-4---interpreter-RINO-GAELICO %
```

Then, I reran with the following commands: `java interpreter.Interpreter "sample_files/simple.x.cod"`



```
~/Documents/CSC413/assignment-4---interpreter-RINO-GAELICO — -zsh
[duccio@Duccios-MacBook-Pro assignment-4---interpreter-RINO-GAELICO % java interpreter.Interpreter "sample_files/scopes.x.cod"
GOTO start<<1>>
[]
LABEL start<<1>>
[]
LIT 0 i                  int i
[0]
LIT 0 j                  int j
[0,0]
GOTO continue<<3>>
[0,0]
LABEL continue<<3>>
[0,0]
LIT 0 m                  int m
[0,0,0]
LIT 3
[0,0,0,3]
ARGS 1
[0,0,0] [3]
CALL f<<2>>              f(3)
[0,0,0] [3]
LABEL f<<2>>
[0,0,0] [3]
LIT 0 j                  int j
[0,0,0] [3,0]
LIT 0 k                  int k
[0,0,0] [3,0,0]
LOAD 0 i                 <load i>
[0,0,0] [3,0,0,3]
LOAD 1 j                 <load j>
[0,0,0] [3,0,0,3,0]
BOP +
[0,0,0] [3,0,0,3]
LOAD 2 k                 <load k>
[0,0,0] [3,0,0,3,0]
BOP +
[0,0,0] [3,0,0,3]
LIT 2
[0,0,0] [3,0,0,3,2]
BOP +
[0,0,0] [3,0,0,5]
RETURN f<<2>>            exit f: 5
[0,0,0,5]
STORE 2 m                m = 5
```



```
[0,0,0,5]
STORE 2 m                m = 5
[0,0,5]
LOAD 1 j                 <load j>
[0,0,5,0]
LOAD 2 m                 <load m>
[0,0,5,0,5]
BOP +
[0,0,5,5]
ARGS 1
[0,0,5] [5]
CALL Write               W(5)
[0,0,5] [5]
LABEL Write
[0,0,5] [5]
LOAD 0 dummyFormal       <load dummyFormal>
[0,0,5] [5,5]
5
WRITE
[0,0,5] [5,5]
RETURN
[0,0,5,5]
STORE 0 i                i = 5
[5,0,5]
POP 3
[]
HALT
[]
duccio@Duccios-MacBook-Pro assignment-4---interpreter-RINO-GAELICO % _
```

Implementation

Bytecode package

The first requirement for this assignment was to implement the Bytecode classes. Those classes had to extend a parent abstract class called Bytecode.
The Bytecode classes needed to contain instance variables that could be used by the objects to function properly. The Bytecode classes had to set up these instance variables but on the other hand, the parent class could not contain any class-specific variable.
To set up the variables keeping an implementation that would benefit from polymorphism and dynamic binding, we had to introduce a generic method init() in the abstract class. This method would take a Vector of Strings as a parameter and it would be implemented with specific behaviors within each child's class. The Vector represented an array of all the arguments passed with the Bytecode itself. Those arguments would be stored in the instance variables of each Bytecode class.

Bytecodes classes

Besides the instance variables and the method init(), each Bytecode class had to implement a method called execute() that had as a parameter an instance of the Virtual Machine class. These methods would have a class-specific implementation depending on the purpose of each Bytecode.
I left this part of the implementation as one of the last things to do because I wanted to have a fully implemented RunTime stack before dive into the concrete behavior of each bytecode.
The most articulated Bytecode was probably the one that involved increasing or decreasing the runtimeStack and the frames pointers. Even for those Bytecodes, the bulk of the code was contained by the RuntimeStack methods, the Bytecode itself simply invoked a method of the Virtual Machine, which then invoked a method of the Runtime Stack object.
Another particular implementation was represented by the Return Bytecode which had to reset the program counter back to the value recorded at the moment the function was called. To do that I used an instance variable from the Virtual Machine class called jump. Jump would store the value of the program counter that it had to go back once the function was returned.
One more implementation that required particular caution and attention was the one involving the reconversion of Addresses. In this case, I had to implement two interfaces that could help me to manipulate the information contained inside the instance variables related to the addresses. I expand more on this topic in a further paragraph.

Dump Bytecode Class

I also had to implement a brand new Bytecode class called DUMP. The class DumpCode itself did not contain a very articulated implementation. The most articulated part was contained in the Virtual Machine class. The Dump would be set on and off according to the argument associated with the bytecode and the Virtual Machine would then print the Bytecode itself with a representation of the runtime stack divided into different frames, only after the execution of that particular bytecode.

To do that, I surrounded the print statements with if conditionals so that it would be possible to verify if the dump was set to on or off before the output would be printed. The Dump itself did not have to be printed so the ON state would be changed only after its execution (when the state of the dump was still OFF); the OFF state instead would be changed before the execution so that no bytecode would be printed after DUMP OFF was found.
Furthermore, each Bytecode had a toString method that allowed a correct representation of the bytecode itself following the instructions provided in the specifications for this assignment. For some of the bytecode classes, the required format included some additional information that had to be included in the toString method.

CodeTable Class

The CodeTable class had a relatively easy implementation since it mostly behaved as a static HashMap container. Once the method init() was called the CodeTable would initialize a HashMap that would associate each Bytecode with its proper className.
Its purpose is to provide a map that could be used by the ByteCodeLoader to obtain the correct Class name for each Bytecode so that it could use a reflective technique to create an instance of that particular Bbytecode Class.
The implementation of this class represented one of the simplest requirements for this assignment.

BytecodeLoader Class

The BytecodeLoader reads the bytecode file line by line, it splits each line into different parts. One of these parts is the bytecode itself which is used to create a bytecode instance. Once the instance is created and all the arguments are stored into instance variables, the BytecodeLoader stores the instance into an Array inside a Program instance.
One more method inside the Bytecode class is used to return the array with all the bytecodes instances once it is ready to be used.

Program Class

The program class's implementation was fairly simple, its job is to store the whole program's bytecode inside an array.
The most challenging part was the implementation of a method for the conversion of symbolic addresses into numeric addresses.

Symbolic addresses into numeric addresses

To be able to convert the symbolic address I had to manipulate the instance variable inside the specific instance for FalseBranch, GoTo, and Call classes. At first, I decided to downcast and check each instance of Bytecode if it was one of the 3 classes mentioned above.
After the professor's suggestion, I opted for the creation of two different interfaces one for those classes that were Target and one for those classes executing the jump. This way I could use a hashmap to associate every label to its array position and also add each branch instance

to a list. Once the program array was full and it was time to convert the symbolic addresses into numeric addresses, I would traverse the list of jumpers and retrieve for each of them the corresponding label's address, and finally store that address in one of their instance variables. This part of the Interpreter implementation required particular attention and a little bit of caution.

RuntimeStack

The RuntimeStack contained most of the code for this assignment.
Most of the methods resemble the method associated with the stack data structure. It has a peek method to check the element at the top of the stack, a push method to add new elements to the stack, and a pop method to remove one element.
Then it has a method to perform a store action, which removes the value on top of the stack and assigns it to an element at a particular position within the stack.
Another method is called load; it loads an element on top of the stack.
The particularity is that the actual stack doesn't use a stack data structure underneath, but it is an array (vector). On the other hand, the data structure that keeps track of the frames uses an actual stack data structure.
The methods that I listed here above did not represent any particular challenge when it came to their implementation, while the dump method was pretty complicated.

Dump method

The dump method's implementation was one of the most challenging parts of the assignment.
I had to make sure that the runtime stack was represented correctly and the frames were properly divided.
To do so I removed all the frame pointers from their stack and added them to a temporary stack so that I could consult them in reverse order. The first element from the temporary stack would be the lower frame pointer in the runtime stack. Then I traversed the Runtime stack itself, which is a list. Any time I encountered an index position that matched with one of the frame pointers I relocated the pointer into the original stack and opened the corresponding pointer.
I had to account for corner cases. For example when the runtime stack was empty or there was only one frame pointer. In these cases a printed an empty bracket with no elements inside.
To represent the stack correctly I used a String Builder class, it allowed me to append the String and modify it with more flexibility. For example, when I had reached the end of a frame, I removed the last comma after the last element.

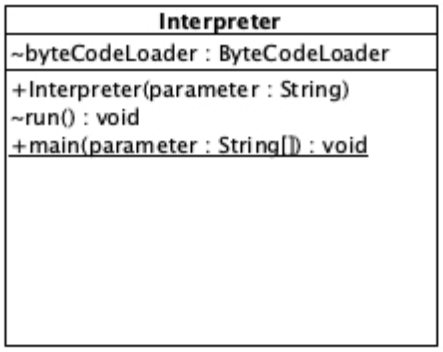
Code Organization

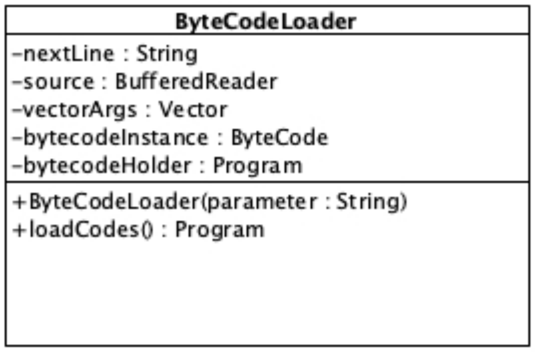
The classes were already neatly organized into packages, I only added the new Bytecodes into the Bytecode package and create a new folder for documentation. I also created a new package for the classes that I implemented to support BOP bytecode.

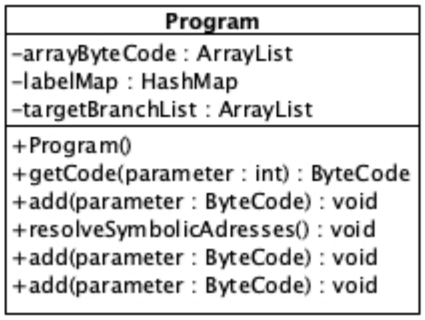
Class Diagram

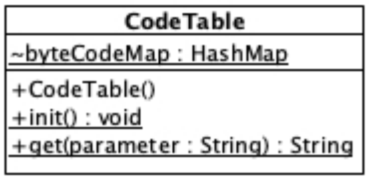
The class diagrams shown below represent all the classes in this project.

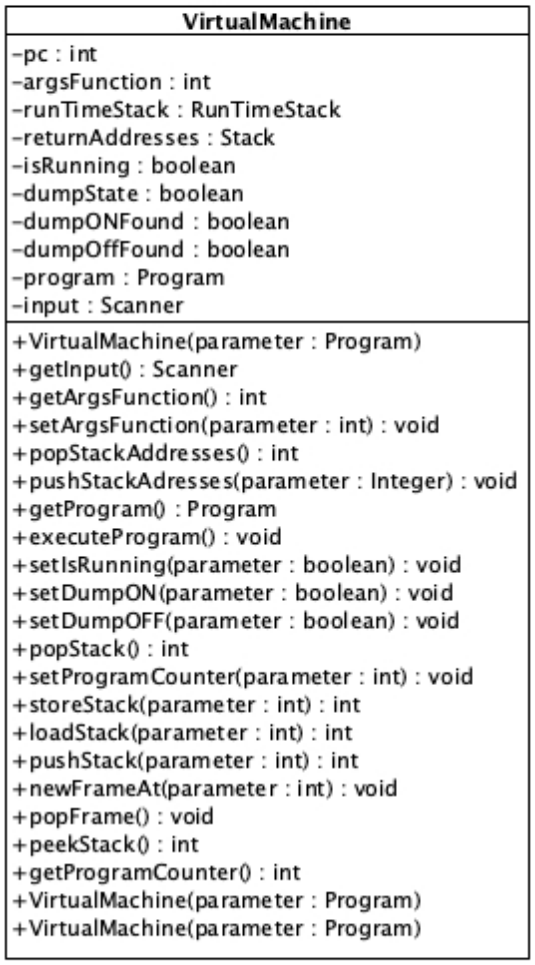
Here are the classes directly contained by the interpreter package:

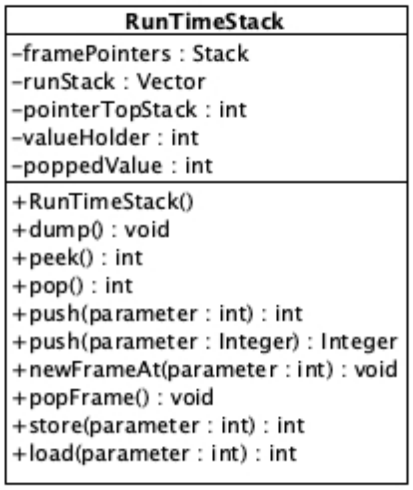

Interpreter
~byteCodeLoader : ByteCodeLoader
+Interpreter(parameter : String)
~run() : void
+main(parameter : String[]) : void



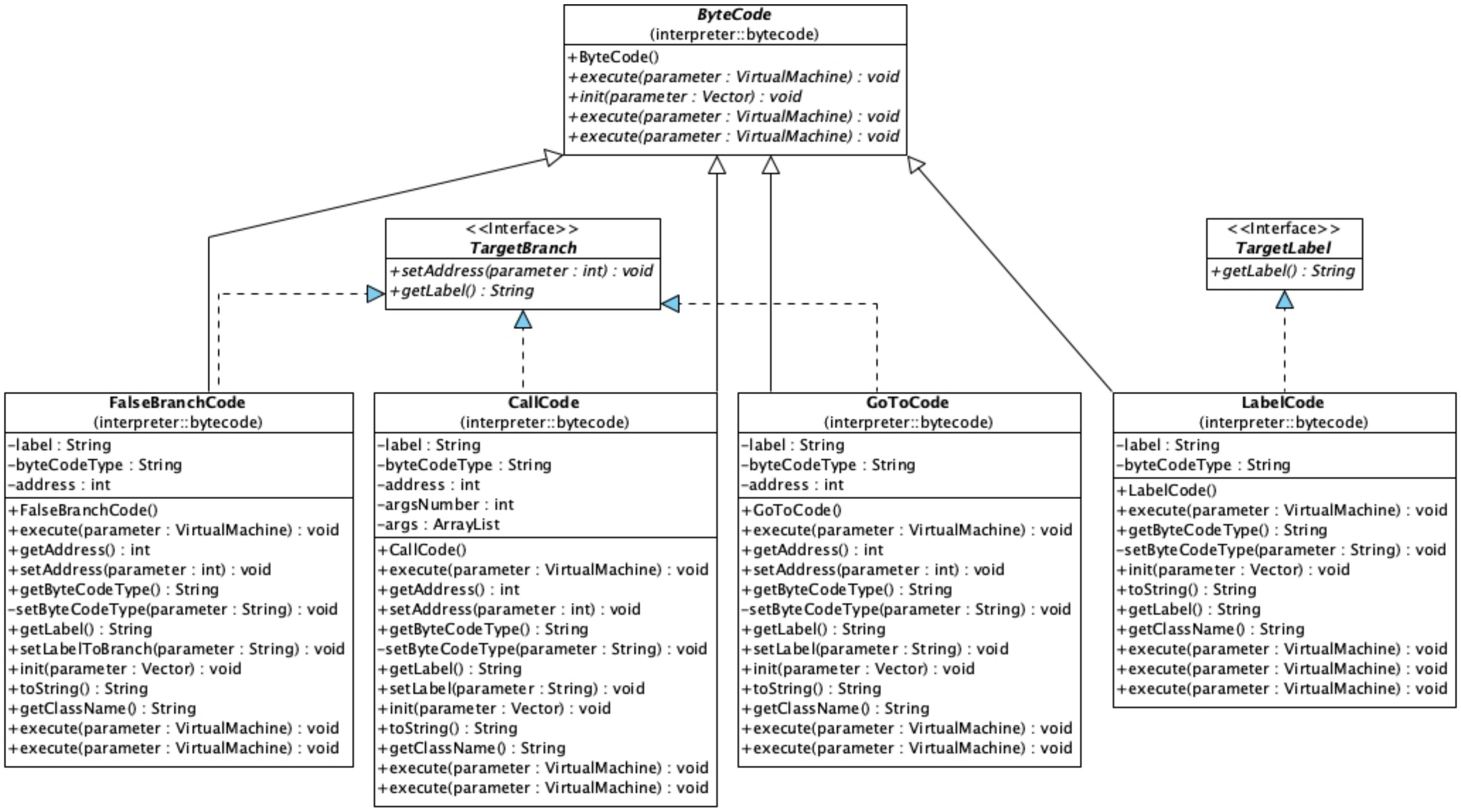

Program
-arrayByteCode : ArrayList
-labelMap : HashMap
-targetBranchList : ArrayList
+Program()
+getCode(parameter : int) : ByteCode
+add(parameter : ByteCode) : void
+resolveSymbolicAdresses() : void
+add(parameter : ByteCode) : void
+add(parameter : ByteCode) : void



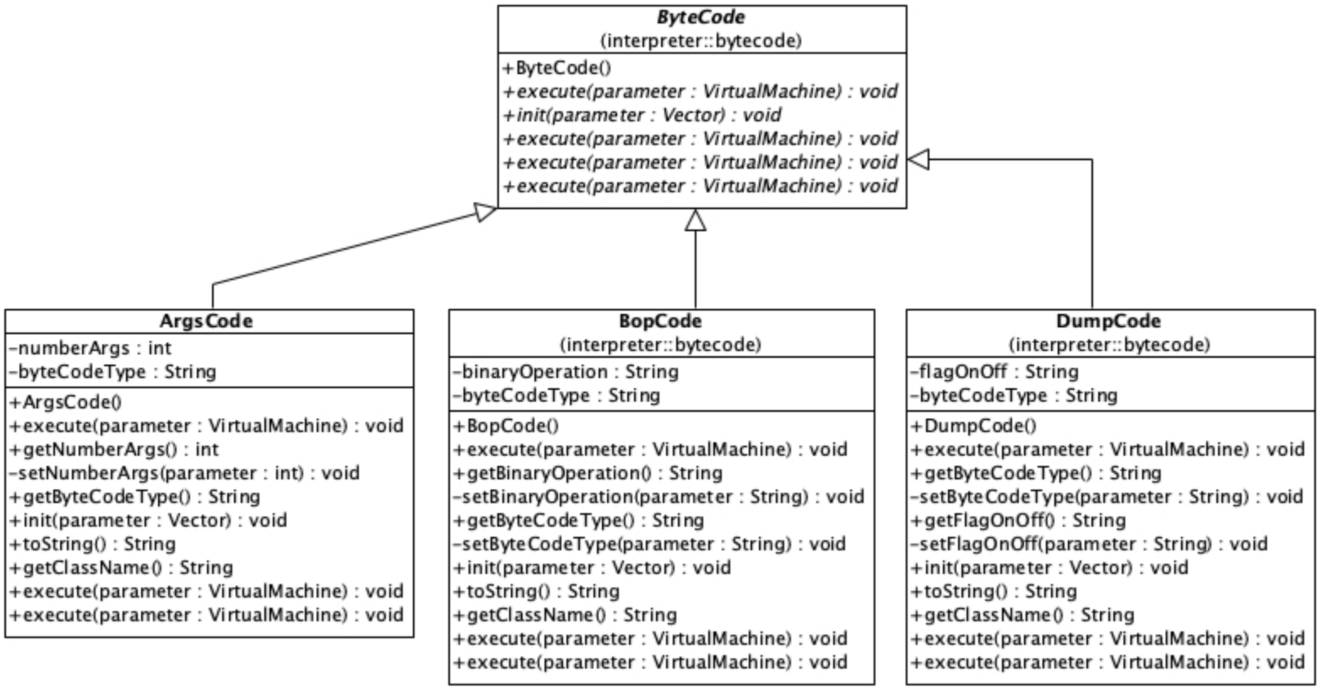

VirtualMachine
-pc : int
-argsFunction : int
-runTimeStack : RunTimeStack
-returnAddresses : Stack
-isRunning : boolean
-dumpState : boolean
-dumpONFound : boolean
-dumpOffFound : boolean
-program : Program
-input : Scanner
+VirtualMachine(parameter : Program)
+getInput() : Scanner
+getArgsFunction() : int
+setArgsFunction(parameter : int) : void
+popStackAddresses() : int
+pushStackAdresses(parameter : Integer) : void
+getProgram() : Program
+executeProgram() : void
+setIsRunning(parameter : boolean) : void
+setDumpON(parameter : boolean) : void
+setDumpOFF(parameter : boolean) : void
+popStack() : int
+setProgramCounter(parameter : int) : void
+storeStack(parameter : int) : int
+loadStack(parameter : int) : int
+pushStack(parameter : int) : int
+newFrameAt(parameter : int) : void
+popFrame() : void
+peekStack() : int
+getProgramCounter() : int
+VirtualMachine(parameter : Program)
+VirtualMachine(parameter : Program)

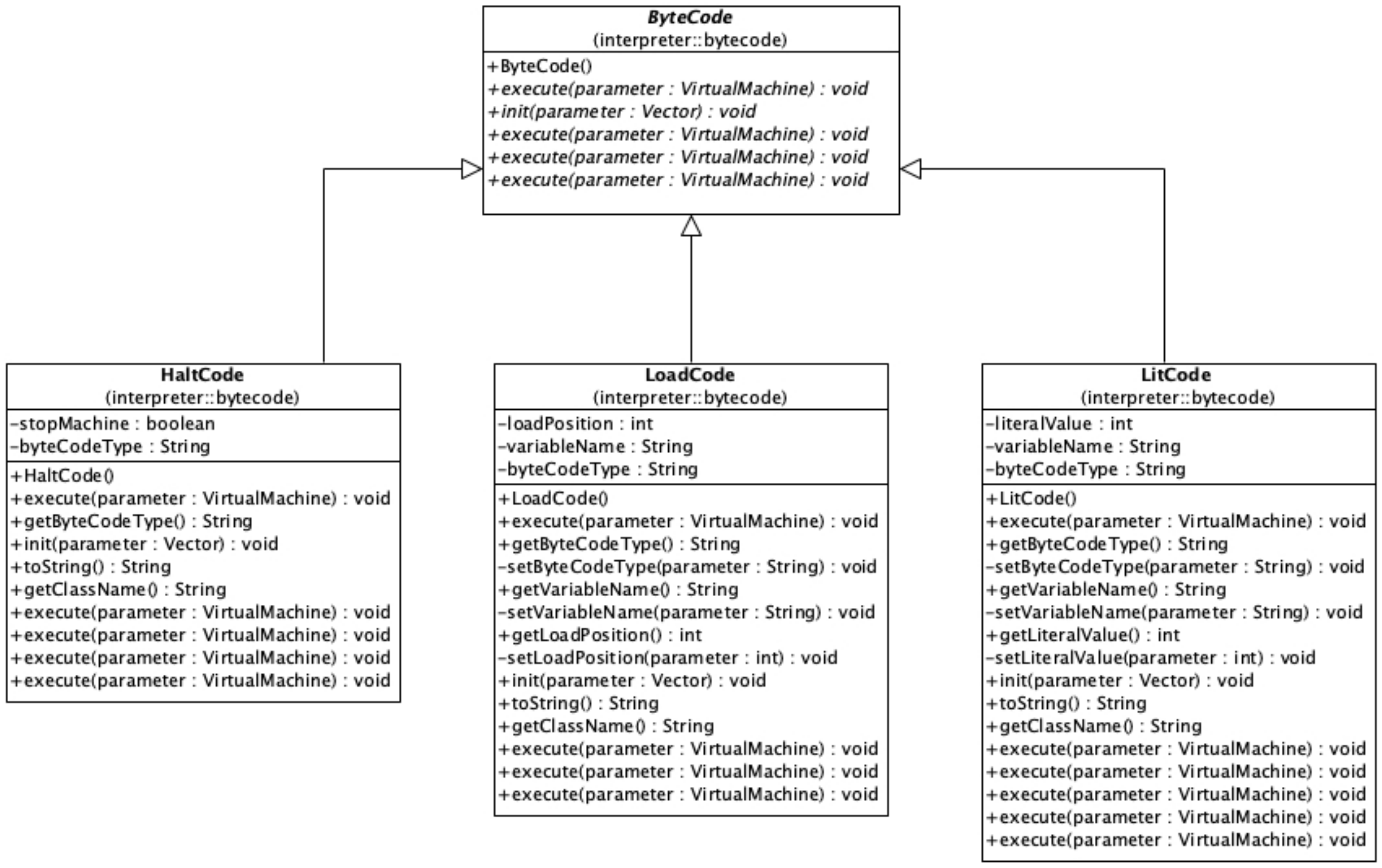


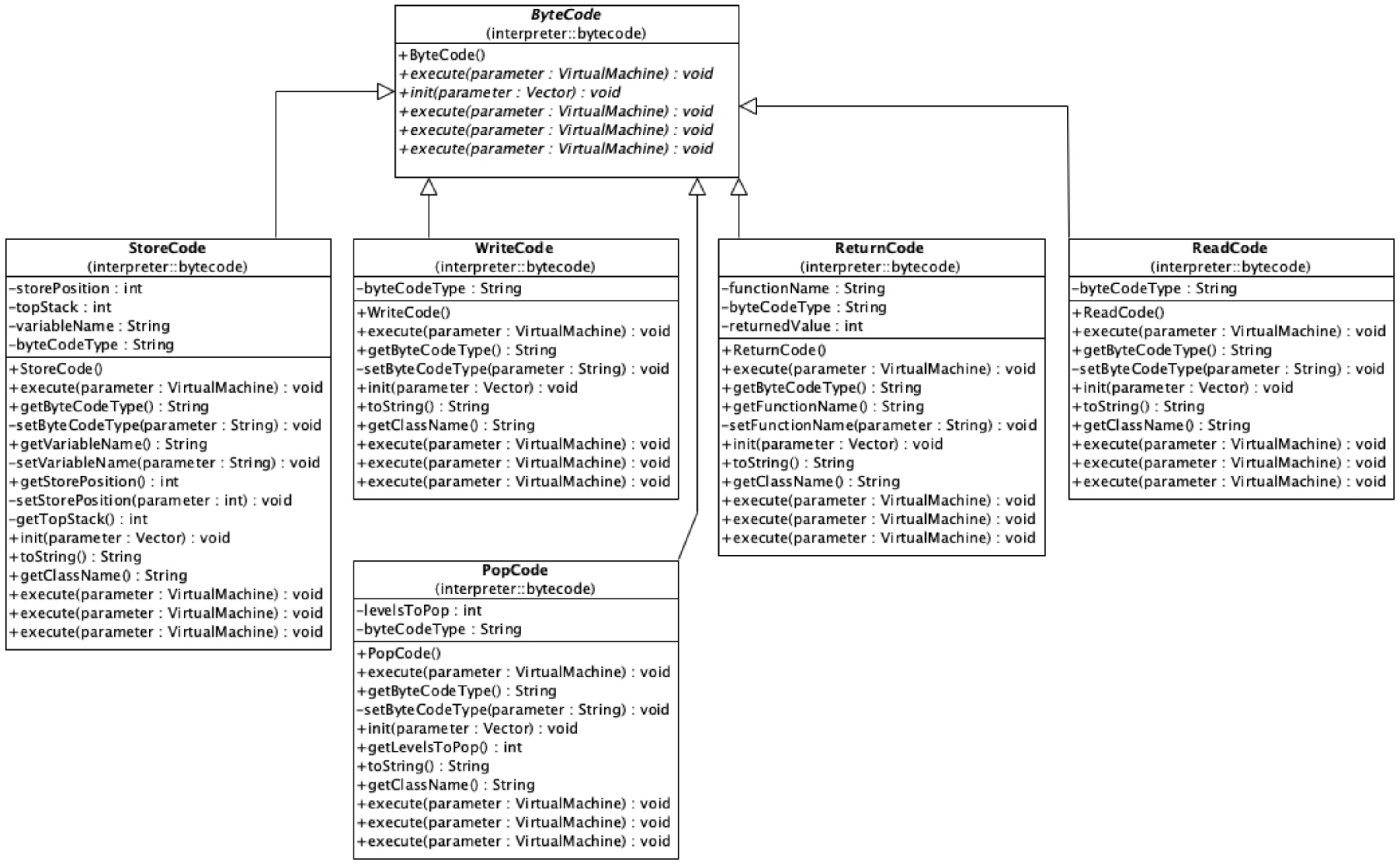
Here is the diagram for the classes that were interested by the conversion of address:


ByteCode
(interpreter::bytecode)
+ByteCode()
+execute(parameter : VirtualMachine) : void
+init(parameter : Vector) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
<<Interface>>
TargetBranch
+setAddress(parameter : int) : void
+getLabel() : String
<<Interface>>
TargetLabel
+getLabel() : String
FalseBranchCode
(interpreter::bytecode)
-label : String
-byteCodeType : String
-address : int
+FalseBranchCode()
+execute(parameter : VirtualMachine) : void
+getAddress() : int
+setAddress(parameter : int) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+getLabel() : String
+setLabelToBranch(parameter : String) : void
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
CallCode
(interpreter::bytecode)
-label : String
-byteCodeType : String
-address : int
-argsNumber : int
-args : ArrayList
+CallCode()
+execute(parameter : VirtualMachine) : void
+getAddress() : int
+setAddress(parameter : int) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+getLabel() : String
+setLabel(parameter : String) : void
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
GoToCode
(interpreter::bytecode)
-label : String
-byteCodeType : String
-address : int
+GoToCode()
+execute(parameter : VirtualMachine) : void
+getAddress() : int
+setAddress(parameter : int) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+getLabel() : String
+setLabel(parameter : String) : void
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
LabelCode
(interpreter::bytecode)
-label : String
-byteCodeType : String
+LabelCode()
+execute(parameter : VirtualMachine) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+init(parameter : Vector) : void
+toString() : String
+getLabel() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void

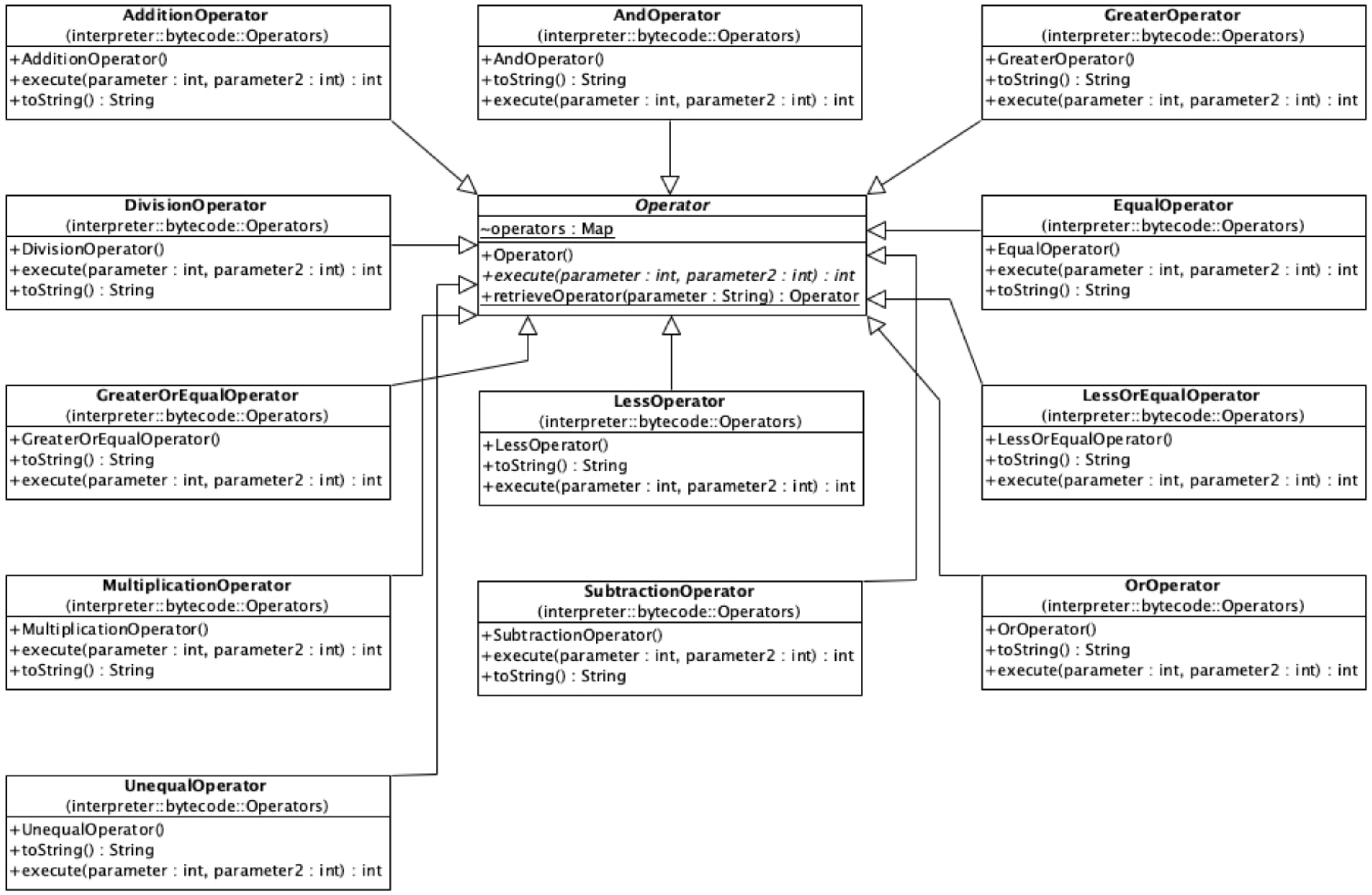
Here is a diagram for the remaining Bytecodes classes that I implemented:


ByteCode
(interpreter::bytecode)
+ByteCode()
+execute(parameter : VirtualMachine) : void
+init(parameter : Vector) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
ArgsCode
-numberArgs : int
-byteCodeType : String
+ArgsCode()
+execute(parameter : VirtualMachine) : void
+getNumberArgs() : int
-setNumberArgs(parameter : int) : void
+getByteCodeType() : String
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
BopCode
(interpreter::bytecode)
-binaryOperation : String
-byteCodeType : String
+BopCode()
+execute(parameter : VirtualMachine) : void
+getBinaryOperation() : String
-setBinaryOperation(parameter : String) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
DumpCode
(interpreter::bytecode)
-flagOnOff : String
-byteCodeType : String
+DumpCode()
+execute(parameter : VirtualMachine) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+getFlagOnOff() : String
-setFlagOnOff(parameter : String) : void
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void


ByteCode
(interpreter::bytecode)
+ByteCode()
+execute(parameter : VirtualMachine) : void
+init(parameter : Vector) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
HaltCode
(interpreter::bytecode)
-stopMachine : boolean
-byteCodeType : String
+HaltCode()
+execute(parameter : VirtualMachine) : void
+getByteCodeType() : String
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
LoadCode
(interpreter::bytecode)
-loadPosition : int
-variableName : String
-byteCodeType : String
+LoadCode()
+execute(parameter : VirtualMachine) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+getVariableName() : String
-setVariableName(parameter : String) : void
+getLoadPosition() : int
-setLoadPosition(parameter : int) : void
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
LitCode
(interpreter::bytecode)
-literalValue : int
-variableName : String
-byteCodeType : String
+LitCode()
+execute(parameter : VirtualMachine) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+getVariableName() : String
-setVariableName(parameter : String) : void
+getLiteralValue() : int
-setLiteralValue(parameter : int) : void
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void


ByteCode
(interpreter::bytecode)
+ByteCode()
+execute(parameter : VirtualMachine) : void
+init(parameter : Vector) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
StoreCode
(interpreter::bytecode)
-storePosition : int
-topStack : int
-variableName : String
-byteCodeType : String
+StoreCode()
+execute(parameter : VirtualMachine) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+getVariableName() : String
-setVariableName(parameter : String) : void
+getStorePosition() : int
-setStorePosition(parameter : int) : void
-getTopStack() : int
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
WriteCode
(interpreter::bytecode)
-byteCodeType : String
+WriteCode()
+execute(parameter : VirtualMachine) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
ReturnCode
(interpreter::bytecode)
-functionName : String
-byteCodeType : String
-returnedValue : int
+ReturnCode()
+execute(parameter : VirtualMachine) : void
+getByteCodeType() : String
+getFunctionName() : String
-setFunctionName(parameter : String) : void
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
ReadCode
(interpreter::bytecode)
-byteCodeType : String
+ReadCode()
+execute(parameter : VirtualMachine) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+init(parameter : Vector) : void
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
PopCode
(interpreter::bytecode)
-levelsToPop : int
-byteCodeType : String
+PopCode()
+execute(parameter : VirtualMachine) : void
+getByteCodeType() : String
-setByteCodeType(parameter : String) : void
+init(parameter : Vector) : void
+getLevelsToPop() : int
+toString() : String
+getClassName() : String
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void
+execute(parameter : VirtualMachine) : void

Here is a diagram for Operator Classes inside Operators package:


AdditionOperator
(interpreter::bytecode::Operators)
+AdditionOperator()
+execute(parameter : int, parameter2 : int) : int
+toString() : String
AndOperator
(interpreter::bytecode::Operators)
+AndOperator()
+toString() : String
+execute(parameter : int, parameter2 : int) : int
GreaterOperator
(interpreter::bytecode::Operators)
+GreaterOperator()
+toString() : String
+execute(parameter : int, parameter2 : int) : int
DivisionOperator
(interpreter::bytecode::Operators)
+DivisionOperator()
+execute(parameter : int, parameter2 : int) : int
+toString() : String
Operator
~operators : Map
+Operator()
+execute(parameter : int, parameter2 : int) : int
+retrieveOperator(parameter : String) : Operator
EqualOperator
(interpreter::bytecode::Operators)
+EqualOperator()
+execute(parameter : int, parameter2 : int) : int
+toString() : String
GreaterOrEqualOperator
(interpreter::bytecode::Operators)
+GreaterOrEqualOperator()
+toString() : String
+execute(parameter : int, parameter2 : int) : int
LessOperator
(interpreter::bytecode::Operators)
+LessOperator()
+toString() : String
+execute(parameter : int, parameter2 : int) : int
LessOrEqualOperator
(interpreter::bytecode::Operators)
+LessOrEqualOperator()
+toString() : String
+execute(parameter : int, parameter2 : int) : int
MultiplicationOperator
(interpreter::bytecode::Operators)
+MultiplicationOperator()
+execute(parameter : int, parameter2 : int) : int
+toString() : String
SubtractionOperator
(interpreter::bytecode::Operators)
+SubtractionOperator()
+execute(parameter : int, parameter2 : int) : int
+toString() : String
OrOperator
(interpreter::bytecode::Operators)
+OrOperator()
+toString() : String
+execute(parameter : int, parameter2 : int) : int
UnequalOperator
(interpreter::bytecode::Operators)
+UnequalOperator()
+toString() : String
+execute(parameter : int, parameter2 : int) : int

Results and Conclusion

This project was particularly interesting because it built on the previous work done with the lexer and the parser. The interpreter represented a natural prosecution after the previous assignments.
This last assignment was probably the most fun of the whole compiler project because it had more variation than the others.
It was very stimulating to work on a bytecode file. It offered a new perspective on how a program works and executes code.

Challenges

The most challenging part of the assignment was the dump method inside Runtime Stack. It required many attempts and every time I thought I found the perfect solution, a new corner case would come up and I had to redesign the whole method. In the end, I found a satisfying solution, even though I don't consider it perfect. It might be optimized a little bit more, especially in the way it treats empty stacks.
Another challenging part was represented by the conversion of symbolic addresses into numeric addresses. My initial solution was not completely satisfying. First, it used downcast, which is considered not a good practice. Secondly, it required many checks and numerous steps to be able to determine exactly which instance was encountered. The professor's tip gave me a chance to rethink my approach and I found his solution way more elegant than mine. That is why I decide to reimplement the conversion method and use interfaces instead of downcasting.
Finally, a challenge was represented by the underflow check. I was not sure how to deal with it. I decided to prevent underflow by outputting a message and simply returning the last element popped in a previous call of the pop method.

Future Work

I would like to be able to fill the gap between the Parser and the Interpreter. Probably that gap is covered by the Codegenerator and the Constrainer classes. It would be interesting to be able to code a compiler from top to bottom.
Also, I would like to implement an interpreter that can handle different primitive data types. At the moment our interpreter only deals with integers but would be interesting to work also with doubles or characters.

Summary of Technical Work

Category	Description		Notes
Code Quality (15)		✓ Code is clean, well formatted (appropriate white space and indentation)	
		✓ Classes, methods, and variables are meaningfully named (no comments exist to explain functionality - the identifiers serve that purpose)	
		✓ Methods are small and serve a single purpose	
		✓ Code is well organized into a meaningful file structure	
Documentation (30)		✓ A PDF is submitted that contains:	
		✓ Full name/Student ID	
		✓ A link to the github repository	
	Overview	✓ Project introduction	
	Overview	✓ Summary of technical work	
	Overview	✓ Execution and development environment described	
	Overview	✓ Scope of work described (including what work was completed)	
		✓ Command line instructions to compile and execute	
		✓ Assumptions made	
	Implementation discussion	✓ Class diagram with hierarchy	
	Implementation discussion	✓ Implementation decisions	
	Implementation discussion	✓ Code organization	
		✓ Results/Conclusions	
		✓ Formatting	
Requirement 1		✓	
Requirement 2		✓	
Requirement 3		✓	